

#2

500.38828X00

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): T. SAKAKIBARA, et al
Serial No.:
Filed: July 28, 2000
Title: COMPUTER SYSTEM
Group:



LETTER CLAIMING RIGHT OF PRIORITY

Honorable Commissioner of
Patents and Trademarks
Washington, D.C. 20231

July 28, 2000

Sir:

Under the provisions of 35 USC 119 and 37 CFR 1.55, the applicant(s) hereby claim(s) the right of priority based on Japanese Patent Application No.(s) 11-216614 filed July 30, 1999.

A certified copy of said Japanese Application is attached.

Respectfully submitted,

ANTONELLI TERRY, STOUT & KRAUS, LLP

A handwritten signature in black ink, appearing to be "Carl I. Brundidge", written over a horizontal line.

Carl I. Brundidge
Registration No. 29,621

CIB/nac
Attachment
(703) 312-6600

日 本 国 特 許 庁
PATENT OFFICE
JAPANESE GOVERNMENT

JC682 U.S. PTO
09/628718
07/28/00

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application:

1999年 7月30日

出 願 番 号
Application Number:

平成11年特許願第216614号

出 願 人
Applicant (s):

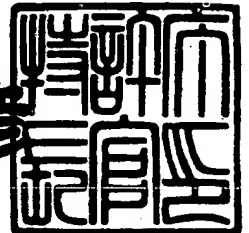
株式会社日立製作所

CERTIFIED COPY OF
PRIORITY DOCUMENT

2000年 6月23日

特許庁長官
Commissioner,
Patent Office

近藤 隆彦



【書類名】 特許願

【整理番号】 K9052331

【提出日】 平成11年 7月30日

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 12/08

【発明者】

【住所又は居所】 神奈川県秦野市堀山下 1 番地 株式会社 日立製作所
エンタープライズサーバ事業部内

【氏名】 榊原 忠幸

【発明者】

【住所又は居所】 神奈川県秦野市堀山下 1 番地 株式会社 日立製作所
エンタープライズサーバ事業部内

【氏名】 大原 功

【発明者】

【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目 2 8 0 番地 株式会社
日立製作所 中央研究所内

【氏名】 明石 英也

【発明者】

【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目 2 8 0 番地 株式会社
日立製作所 中央研究所内

【氏名】 對馬 雄次

【発明者】

【住所又は居所】 神奈川県横浜市戸塚区吉田町 2 9 2 番地 株式会社 日
立製作所 生産技術研究所内

【氏名】 村岡 諭

【特許出願人】

【識別番号】 000005108

【氏名又は名称】 株式会社 日立製作所

【代表者】 庄山 悦彦

【代理人】

【識別番号】 100073760

【弁理士】

【氏名又は名称】 鈴木 誠

【手数料の表示】

【予納台帳番号】 011800

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 コンピュータシステム

【特許請求の範囲】

【請求項 1】 CPUと、メモリと、前記CPUと前記メモリの間の階層に位置するキャッシュとを具備するコンピュータシステムにおいて、

前記CPUからのリクエストが前記キャッシュにヒットするかを判断して、前記キャッシュあるいは前記メモリに対してリクエストを発行するコヒーレントコントローラと、前記コヒーレントコントローラからのリクエストに従って前記キャッシュに登録されているデータの読み書きを制御するキャッシュデータコントローラとを備え、

前記コヒーレントコントローラは、前記CPUからリードリクエストを受け付けると、前記キャッシュのヒット判定を行うとともに前記キャッシュデータコントローラに投棄リードリクエストを発行し、ヒット判定がキャッシュヒットの場合には前記キャッシュデータコントローラにリードリクエストを発行する手段を有し、

前記キャッシュデータコントローラは、前記コヒーレントコントローラからの投棄リードリクエストを受け付けると、前記キャッシュからデータを読み出して保持する手段と、前記コヒーレントコントローラからリードリクエストを受け付けると、前記保持しておいた投棄リードデータを前記CPUへのリードリプレイデータとして送出する手段を有する、
ことを特徴とするコンピュータシステム。

【請求項 2】 請求項 1 記載のコンピュータシステムにおいて、

前記コヒーレントコントローラは、ヒット判定がキャッシュミスの場合には、キャッシュデータコントローラに投棄リードデータ破棄リクエストを発行する手段を有し、

前記キャッシュデータコントローラは、前記コヒーレントコントローラから投棄リードデータ破棄リクエストを受け付けると、当該投棄リードデータ破棄リクエストに対応する投棄リードリクエストの投棄リードデータを破棄する手段を有する、

ことを特徴とするコンピュータシステム。

【請求項 3】 請求項 1 もしくは 2 記載のコンピュータシステムにおいて、キャッシュは n ウエイアソシアティブキャッシュからなり、キャッシュデータコントローラは、コヒーレントコントローラから n ウエイ分の投棄リードリクエストを受け付けて、キャッシュから n ウエイ分のデータを読み出して保持することを特徴とするコンピュータシステム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、コンピュータシステムに係り、特に CPU とメインメモリの間にキャッシュを具備するコンピュータシステムにおいて、キャッシュデータの投棄的リード制御方式に関する。

【0002】

【従来の技術】

近年、CPU の性能向上はメモリの性能向上に比べて遥かに高くその性能差はますます広がる方向にある。メインメモリより高速でメインメモリの内容の一部を記憶するキャッシュは、このような CPU とメモリの性能差を吸収し、実効メモリアクセス時間を短縮するために使用される。

【0003】

コンピュータシステムにおけるキャッシュの容量の選択は、そのコンピュータシステムの構成に大きく依存する。高性能な CPU には、それ自体が大容量なキャッシュを持つ場合が多い。このような CPU を数多く接続する場合、2 ～ 8 程度の CPU をバスまたはスイッチで接続し、さらにそれ以上の CPU を接続する場合には、それらを又、バスまたはスイッチで接続するといった多階層のメモリシステムを構成することが多い。このような構成をとると、CPU とメインメモリの間のアクセスレイテンシが増加し、CPU でキャッシュミスが生じた場合の性能に大きな影響を及ぼす。このため、最も上の階層の 2 ～ 8 程度の CPU を接続したバスあるいはスイッチ制御の階層にキャッシュを設けて、CPU のキャッシュあふれが生じた時の性能低下を防ぐ必要がある。例えば、特開平 9 - 1 2 8

346号公報には、このような階層バスシステムのキャッシュ構成例が開示されている。このときのキャッシュ容量は、その上につながる全てのCPUの総キャッシュ容量以上である必要がある。これはもしCPUキャッシュと同じあるいは小さいキャッシュ容量しか持たない場合には、CPUキャッシュであふれが生じた時に、その下の階層でもキャッシュあふれが生じやすくなり、著しいシステム性能の低下を招く恐れがあるためである。

【0004】

ところで、キャッシュの高速アクセスを実現するために、キャッシュにSRAM等の高速な素子を用いることが一般的である。通常、このSRAM内の同一ロケーションにキャッシュタグ（タグアドレス）とキャッシュデータを格納して、リードリクエストを処理する場合には、キャッシュタグとキャッシュデータを同時に読み出し、リクエストアドレスでキャッシュタグをチェックして、ヒットしていれば直ちにキャッシュデータを使うことができるように構成する。しかし、SRAMはメインメモリ等に使用されるDRAMと比較して集積度が一桁以上低く、大容量キャッシュを構成するにはSRAMを多数用いる必要がある。SRAMを多数用いる場合、多数のSRAMとのインタフェースをとるため、キャッシュを制御するLSIのピン数が増加して一つのLSIで構成できなくなる。キャッシュタグ部分はキャッシュヒットチェックに使用し、このヒットチェックにかかる時間の増加は、直接メモリアクセスレイテンシの増加につながるため、キャッシュタグ部分とのインタフェースを持つLSIはCPUバスと同一LSIに入れる必要がある。キャッシュデータ部分とのインタフェースを持つLSIはCPUバスを含むLSIと異なるLSIとし、そのインタフェースをCPUバス程度のデータ幅とすることによって、LSIのピンネックを解消することができる。

【0005】

一方、キャッシュのヒット率を向上させる方式として、セットアソシアティブ方式がある。例えば、特開平5-225053号公報にセットアソシアティブキャッシュのタグ比較を行う方式およびその高速化について開示されている。

【0006】

セットアソシアティブ方式におけるヒットチェックでは、複数ウェイの複数キ

キャッシュラインのキャッシュタグを読み出して、複数ライン同時にヒットチェックを行う。このとき、複数ラインのどのデータを使用するかはキャッシュヒットチェックが完了するまでわからない。CPU内部に搭載されているキャッシュ（オンチップキャッシュ）では、キャッシュデータの読み出しをキャッシュタグの読み出しと同時にを行い、キャッシュヒットチェックが完了した後で必要とするデータだけを選択することによってキャッシュアクセスレイテンシを削減する方式をとることが一般的である。

【0007】

図12に、この種のセットアソシアティブキャッシュの構成例を示す。図12はNエントリからなる4ウェイセットアソシアティブキャッシュを示している。各エントリは4つのウェイ、第0ウェイ1000、第1ウェイ1001、第2ウェイ1002、第3ウェイ1003からなる。キャッシュに含まれる情報は、キャッシュの状態（有効、無効）を示すSTAT1004、キャッシュタグ（アドレス）1005、キャッシュデータ1006からなる。一般にキャッシュのエントリ番号はメモリアドレスの下位アドレスを用い、上位アドレスをキャッシュタグとする方法がとられる。オンチップキャッシュでは、図12の様にキャッシュタグ1005とキャッシュデータ1006をあわせて格納してあるため、該当エントリの各ウェイ1000～1003のキャッシュタグ1005とキャッシュデータ1006を同時に読み出して、キャッシュヒットしたウェイ番号を使って直ちにデータを選択することができる。

【0008】

しかし、大容量セットアソシアティブキャッシュを実現しようとする、キャッシュデータとインタフェースを持つLSIと、CPUバスとキャッシュタグとのインタフェースを含むLSIとを分離せざるをえず、この場合にはキャッシュタグとキャッシュデータを同時に読み出すことはできない。そこで、キャッシュタグとキャッシュデータを別々に読み出すことになるが、このとき、物理的な制約によってキャッシュタグとインタフェースを持つLSIとキャッシュデータとインタフェースを持つLSIとの間のデータ幅がCPUバス程度しかない場合、複数ライン分のキャッシュデータを全てCPUバス側のLSIまで読み出すには

時間がかかり過ぎてしまう。例えば、CPUバス幅が8バイトで、ラインサイズが32バイトの場合、4ウエイ分のラインをキャッシュデータ側LSIからキャッシュタグ側LSIに転送するのに4サイクル×4ウエイ=16サイクルかかってしまう。これはキャッシュを引く度に16サイクルかかってしまうことを意味しており、著しく性能が低下してしまう。この性能低下を防ぐには、キャッシュヒットチェック結果が分かってからキャッシュデータを読み出す必要が出てくるが、これではキャッシュのアクセスレイテンシの増大を引き起こすことになってしまう。

【0009】

【発明が解決しようとする課題】

上記のように、セットアソシアティブ方式等の大容量キャッシュをCPUとメインメモリの間に設ける場合、LSIのピン数などの制約からキャッシュタグ部分とキャッシュデータ部分を別々のLSIに分けて管理する必要性が出てくる。そのような構成をとった場合、キャッシュタグを読み出してキャッシュヒットチェックを行ってからキャッシュデータを読み出すとキャッシュ読み出しレイテンシが増大するという問題があった。

【0010】

本発明の目的は、CPUとメインメモリの間の階層に位置するnウエイセットアソシアティブキャッシュ等のキャッシュを具備するコンピュータシステムにおいて、上記のようにキャッシュタグ部分とキャッシュデータ部分を別LSIで管理した場合のキャッシュデータの読み出し時間の短縮を実現することにある。

【0011】

【課題を解決するための手段】

上記目的を達成するため、本発明では、キャッシュヒットチェックを行う前に、キャッシュデータ部分のコントローラに投棄的なリードリクエストを発行して、キャッシュからのデータを先読みして保持しておき、キャッシュヒットチェックが終了した時点で、キャッシュヒットした場合、投棄的にリードしておいたデータを使用することによって、キャッシュヒットチェック時間を短縮してキャッシュデータを読み出すようにしたものである。

【0 0 1 2】

【発明の実施の形態】

以下、本発明の一実施の形態について図面により説明する。

図 1 に、本発明の一実施の形態のコンピュータシステムを示す。本システムは、2つのCPU (0) 1、CPU (1) 2、ストレージコントローラ (SCU) 4、キャッシュタグ部 5、キャッシュデータコントローラ 6、キャッシュデータ部 7、メインメモリ 8、CPU (0) 1 と CPU (1) 2 と SCU 4 を接続するバス 3 などから構成される。さらに SCU 4 はバス 1 6、メモリアクセスリクエストキュー 1 7、ライトデータバッファ 1 8、リードリプライデータバッファ 1 9、コヒーレントコントローラ 2 0、メモリアクセスコントローラ 2 1、バス 2 2 と、それらを接続するバス 2 3 ~ 2 9 から構成される。なお、ここでは CPU の数 (ノード数) は 2 つであるが、勿論、それ以上のノード数でも、また、1 つでもかまわない。

【0 0 1 3】

図示しないが、CPU (0) 1 と CPU (1) 2 には内臓キャッシュが設けられているものと仮定する。また、キャッシュタグ部 5 とキャッシュデータ部 7 は高速なメモリ素子である SRAM などで構成され、メインメモリ 8 は低速なメモリ素子である DRAM などで構成されているものとする。さらに、キャッシュタグ部 5 およびキャッシュデータ部 7 で構成される本システムのキャッシュは、4ウェイセットアソシアティブキャッシュであるものとする。

【0 0 1 4】

図 2 に、CPU (0) 1 および CPU (1) 2 からのリクエストアドレスとキャッシュタグ、キャッシュエントリ番号の関係を示す。本実施形態では、CPU (0) 1 および CPU (1) 2 からのリクエストアドレスは 3 2 ビットであり、キャッシュエントリ数は 2 5 6 K エントリであるとする。また、キャッシュラインサイズは 3 2 バイトであると仮定する。図 2 において、1 0 0 は CPU (0) 1 および CPU (1) 2 から出力されるリクエストアドレス ($ADR < 31 : 0 >$) を示す。キャッシュラインサイズは 3 2 バイトであるため、 $ADR 100$ の下位 6 ビットはキャッシュライン内のアドレスを示すことになる。キャッシュエ

ントリ数は 2 5 6 K エントリであるため、 $ADR < 24 : 7 >$ の 1 8 ビットがキャッシュエントリ番号 1 0 2 となる。そして、残りの上位アドレスである $ADR < 31 : 25 >$ がキャッシュタグ 1 0 1 となる。

【 0 0 1 5 】

図 3 および図 4 にキャッシュタグ部 5 とキャッシュデータ部 7 の構成例を示す。図 3 に示すように、キャッシュタグ部 5 は 2 5 6 K エントリ、4 ウエイのキャッシュ状態 (STAT) 2 1 0 とキャッシュタグ 2 1 1 の集合体である。また、図 4 に示すように、キャッシュデータ部 7 は 2 5 6 K エントリ、4 ウエイのキャッシュデータ 2 2 0 の集合体である。キャッシュタグ部 5 とキャッシュデータ部 7 の各エントリ、各ウエイは一対一に対応し、例えば、キャッシュデータ部 7 の 0 エントリ、0 ウエイのブロックに格納されたキャッシュデータに対応するキャッシュタグは、キャッシュタグ部 5 の 0 エントリ、0 ウエイのブロックに格納される。キャッシュ状態 (STAT) 2 1 0 は、当該ブロックのキャッシュデータ (キャッシュライン) の有効、無効を示す。

【 0 0 1 6 】

図 1 に戻り、キャッシュデータコントローラ 6 は、SCU 4 からのキャッシュデータのリード/ライトリクエストを受け付けて、キャッシュデータ部 7 に対するキャッシュデータの読み書きを行う。パス 1 2 は SCU 4 からのアクセスリクエストをキャッシュデータコントローラ 6 に送出するためのバスであり、パス 1 3 は SCU 4 とキャッシュデータコントローラ 6 との間でデータをやりとりするためのバスである。パス 3 0 はキャッシュデータコントローラ 6 がキャッシュデータ部 7 をリード/ライト制御するための信号線であり、パス 3 1 はキャッシュデータコントローラ 6 とキャッシュデータ部 7 との間でデータをやり取りするためのバスである。本実施形態では、4 ウエイセットアソシアティブキャッシュであるため、パス 3 0、3 1 の信号数が大きく、SCU 4 から直接キャッシュデータ部 7 にピンをはることが物理的に不可能なため、データキャッシュコントローラ 6 を SCU 4 とは別チップとして構成しているものとする。これにより、パス 1 2 は、1 3 はそれぞれパス 3 0、3 1 に比較して少ない信号数としている。

【0017】

SCU4内のメモリアクセスリクエストキュー17はCPU(0)1、CPU(1)2からのメモリアクセスリクエストをバッファリングして、コヒーレントコントローラ20がビジーでない場合にメモリアクセスリクエストをコヒーレントコントローラ20に送るためのキューである。データバッファ18はCPU(0)1、CPU(1)2からのライトデータを一時的に格納するバッファであり、データバッファ19はCPU(0)1、CPU(1)2に返すリードリプライデータを一時的に格納するためのバッファである。コヒーレントコントローラ20は、CPU(0)1、CPU(1)2からのメモリアクセスリクエストがキャッシュヒットするかどうかを判定して、キャッシュデータコントローラ6とメモリアクセスコントローラ21にアクセスリクエストを発行する。メモリアクセスコントローラ21は、コヒーレントコントローラ20からのアクセスリクエストに従って、メインメモリ8のアクセス制御を行う。

【0018】

ここで、コヒーレントコントローラ20では、図2に示したようなCPU(0)1およびCPU(1)2からのリクエストアドレス100を分解し、キャッシュエントリ番号102を使って図3に示すキャッシュタグ部5の該当するエントリの各ウエイからキャッシュタグ211を読み出し、これらとリクエストアドレス100のキャッシュタグ101と比較することによってキャッシュヒットチェックを行う。このキャッシュヒットチェック自体は従来と基本的に同様である。

【0019】

次に、図1のコンピュータシステムにおいて一実施の形態の動作を説明する。

CPU(0)1、CPU(1)2では、命令実行に必要なデータが内臓キャッシュにない場合、メモリアクセスリクエストをバス3を介してSCU4に発行する。SCU4では、バス16を介してメモリアクセスリクエストキュー17に該メモリアクセスリクエストを格納する。ライトリクエストの場合にはCPU(0)1、CPU(1)2からデータも送出されるため、SCU4ではライトデータをバス16を介してデータバッファ18に格納する。コヒーレントコントローラ20がビジーでない場合、メモリアクセスリクエストキュー17からコヒーレン

トコントローラ 20 にメモリアクセスリクエストを送出する。

【0020】

コヒーレントコントローラ 20 では、キャッシュタグ部 5 を参照して、受け取ったメモリアクセスリクエストがキャッシュにヒットするかどうかを判定するが、リードリクエストの場合には、キャッシュヒット判定結果を待ってから、キャッシュデータコントローラ 6 を介して、データキャッシュ 7 からデータを読み出すと、アクセスレイテンシが大きくなってしまう。そこで、コヒーレントコントローラ 20 は、キャッシュタグ部 5 によるキャッシュヒット判定を行う前に、キャッシュデータコントローラ 6 に対して投棄的なリードを行うリクエストを発行し、ヒットした場合に読み出す必要のあるデータをあらかじめキャッシュデータ部 7 からキャッシュデータコントローラ 6 まで読み出しておいて、ヒットした場合にはこの先読みしたデータを使用する。

【0021】

図 5 は、コヒーレントコントローラ 20 の一実施の形態の処理フローである。以下に、上記コヒーレントコントローラ 20 の詳細な動作を図 5 を用いて説明する。

【0022】

コヒーレントコントローラ 20 では、CPU (0) あるいは CPU (1) 2 からのメモリアクセスリクエストを受け付けると (ステップ 300)、そのリクエストがリードかどうかを判定する (ステップ 301)。もしリードリクエストであれば、バス 25, 12 を介してキャッシュデータコントローラ 6 に投棄リードリクエストを発行し (ステップ 302)、同時にキャッシュタグ部 5 にバス 23, 10 を介して当該リードリクエストのキャッシュエントリ番号を送り、キャッシュタグ部 5 からバス 11, 24 を介して当該エントリの 4 ウエイ分のキャッシュタグの読み出しを行う (ステップ 303)。このキャッシュタグ部 5 から読み出したキャッシュタグがリードリクエストのキャッシュタグとヒットするかどうかを判定し (ステップ 304)、ヒットした場合には、バス 25, 12 を介してキャッシュデータコントローラ 6 にリードリクエストを発行する (ステップ 305)。このときのリードリクエストには、キャッシュエントリ番号とともにヒッ

トしたウエイ番号が含まれる。キャッシュミスした場合には、メモリアクセスコントローラ 2 1 にリードリクエストを発行し（ステップ 3 0 6）、バス 2 3, 1 0 を介してキャッシュタグ部 5 の該当エントリの所望ウエイに該メモリアクセスリクエストのキャッシュタグを新しく登録する（ステップ 3 0 7）。メモリアクセスコントローラ 2 1 はバス 2 7, 1 4 を介してメインメモリ 8 をアクセスし、バス 1 5, 2 8 にデータを読み出す。このメインメモリ 8 からのリプライデータが帰ってきたら、コヒーレントコントローラ 2 0 では、バス 2 5, 1 2 を介し、キャッシュデータコントローラ 6 に対して、このリプライデータをキャッシュデータ部 7 に登録するライトリクエストを発行し、バス 2 2、バス 2 6, 1 3 を介し、当該リプライデータをライトデータとしてキャッシュデータコントローラ 6 に送出する（ステップ 1 0 8）。同時に、リプライデータを CPU に送るために、バス 2 2 よりデータバッファ 1 9 に当該リプライデータを格納する（ステップ 3 0 9）。

【 0 0 2 3 】

CPU (0) 1 あるいは CPU (1) 2 から受け取ったリクエストがライトリクエストであった場合、コヒーレントコントローラ 2 0 では、リードリクエストの場合と同様にしてキャッシュタグ部 5 から当該エントリの 4 ウエイ分のキャッシュタグを読み出し（ステップ 3 1 0）、キャッシュヒットしているかどうかを判定する（ステップ 3 1 1）。ヒットしていた場合、バス 2 5, 1 2 を介してキャッシュデータコントローラ 6 にライトリクエストを発行し（ステップ 3 1 2）、同時にデータバッファ 1 8 からバス 2 2、バス 2 6, 1 3 を介してライトデータをキャッシュデータコントローラ 6 に送出する（ステップ 3 1 3）。このときのライトリクエストには、キャッシュエントリ番号とともにヒットしたウエイ番号が含まれる。ミスしていた場合には、メモリアクセスコントローラ 2 1 にライトリクエストを発行し（ステップ 3 1 4）、同時にデータバッファ 1 8 からバス 2 2、バス 2 8, 1 5 を介してメインメモリ 8 にライトデータを送出する（ステップ 3 1 5）。メモリアクセスコントローラ 2 1 では、バス 2 7, 1 4 を介してメインメモリ 8 をアクセスし、メインメモリ 8 にデータを書き込む。

【0024】

このように、コヒーレントコントローラ20では、特にリードリクエストを受け付けた場合に、キャッシュタグ部5によるキャッシュヒットチェックを行う前にキャッシュデータコントローラ6に投棄的リードリクエストを発行する機能を持つ。ライトリクエストの場合には基本的に従来と同様である。

【0025】

次に、キャッシュデータコントローラ6の構成および動作を説明する。図1において、キャッシュデータコントローラ6はバス12を介してコヒーレントコントローラ20からの投棄的リードリクエスト、リードリクエスト、ライトリクエストに従って、キャッシュデータ部7とデータのやりとりを行う。

【0026】

図6は、キャッシュデータコントローラ6の詳細なブロック図である。キャッシュデータコントローラ6は、リクエストコントローラ400、投棄リードリクエストバッファ401、アドレスコンパレータ部402、投棄リードデータバッファ403～406、バス407、408～411、セクタ412、413～416、バス417～428からなる。

【0027】

リクエストコントローラ400では、バス12を介してコヒーレントコントローラ20から受け取ったリクエストをデコードし、該受け付けたリクエストの種類によって当該キャッシュデータコントローラ6で行う処理を決定し、各部を制御する。投棄リードリクエストバッファ401は、コヒーレントコントローラ20から受け取った投棄リードリクエストを保持するバッファである。投棄リードデータバッファ403～406は投棄リードリクエストに従ってキャッシュデータ部7から読み出したデータを保持するバッファである。図4に示した通り、本実施形態のキャッシュデータ部7は4ウェイセットアソシアティブであり、第0ウェイのデータは投棄リードデータバッファ403に、第1ウェイのデータは404に、第2ウェイのデータは405に、第3ウェイのデータは406に格納する。アドレスコンパレータ部402は、コヒーレントコントローラ20から受け取った投棄リードリクエスト、リードリクエスト・ライトリクエストが、投棄リ

ードリクエストバッファ401に格納されたリクエストと同一キャッシュエントリであるかどうかをチェックする。

【0028】

図7および図8に投棄リードリクエストバッファ401と投棄リードデータバッファ403～406の構成例を示す。図7に示すように、投棄リードリクエストバッファ401は複数のエントリからなり、各エントリはバリッドビット(V)500、キャッシュエントリ番号501からなる。バリッドビット500はそのエントリが有効・無効であることを示すビットであり、キャッシュエントリ番号501は当該エントリに格納されている投棄リードリクエストが対象とするキャッシュエントリ番号である。投棄リードデータバッファ403～406も、図8に示すように複数のエントリからなり、各エントリには、投棄リードリクエストによりキャッシュデータ部7から投棄的に読み出したキャッシュデータ(32B)600が格納されている。

【0029】

投棄リードリクエストバッファ401のエントリと投棄リードデータバッファ403～406のエントリは一対一に対応する。例えば、投棄リードリクエストバッファ401の0エントリに、ある投棄リードリクエストのキャッシュエントリ番号が格納されているとすると、当該投棄リードリクエストによりキャッシュデータ部7から投棄的に読み出された4ウェイ分キャッシュデータは、投棄リードデータバッファ403～406の0エントリに格納される。なお、これら投棄リードリクエストバッファ401と投棄リードデータバッファ403～406のエントリ数(m)は任意でよい。また、これらバッファ401, 403～406は一体的に構成してもよい。

【0030】

図9はリクエストコントローラ400の一実施形態の処理フローである。以下に、図9により、リクエストコントローラ400を中心にキャッシュデータコントローラ6の詳細な動作を説明する。

【0031】

リクエストコントローラ400では、コヒーレントコントローラ20からパス

12、417を通してリクエストを受け取ると(ステップ700)、まず、投棄リードリクエストかどうかをチェックする(ステップ701)。投棄リードリクエストの場合、投棄リードリクエストバッファ401に同一キャッシュエントリへのリクエストが格納されているかチェックする(ステップ702)。具体的には、リクエストコントローラ400は、投棄リードリクエストのキャッシュエントリ番号をバス419に出力するとともに、投棄リードリクエストバッファ401の各エントリのキャッシュエントリ番号を読み出し、両者をアドレスコンパレータ部402で比較し、比較結果をバス420で受け取ることで、投棄リードリクエストバッファ401に投棄リードリクエストと同一キャッシュエントリが格納されているかチェックする。ここで、もし格納されていれば、新しく受け取った投棄リードリクエストを無視する。もしも投棄リードリクエストバッファ401に同一キャッシュエントリへのリクエストが格納されていない場合、投棄リードリクエストバッファ401がフルかどうかをチェックする(ステップ703)。もしフルでなければ、バス28を通して、投棄リードリクエストバッファ401内の空きエントリに新しい投棄リードリクエストを登録し(ステップ705)、もしフルであれば、投棄リードリクエストバッファ401内の最も古いエントリを無効化した後(ステップ704)、新しいリクエストを登録する。なお、この種の無効化のアルゴリズムはLRU(Least Recentry Used)法として良く知られており、詳しい説明は省略する。この登録した投棄リードリクエストを、バス218、30を介してキャッシュデータ部7にリードリクエストとして転送し、キャッシュデータ部7の該当キャッシュエントリから4ウェイ分のキャッシュデータを読み出し(ステップ706)、バス31、バス408~411、バス423~426を介し、投棄リードデータバッファ403~406内の、投棄リードリクエストバッファ401に投棄リードリクエストを登録したエントリと対応するエントリに、新しく該キャッシュデータを格納する(ステップ707)。この結果、投棄リードリクエストバッファ401がフルの場合には、該投棄リードリクエストバッファ401の無効化したエントリに対応するところの、投棄リードデータバッファ403~406の該当エントリに新しいキャッシュデータが上書きされて格納されることになる。

【 0 0 3 2 】

コヒーレントコントローラ 2 0 から受け取ったリクエストが投棄リードリクエストでなくリードリクエストであった場合には（ステップ 7 0 8）、まず、投棄リードリクエストバッファ 4 0 1 にリードリクエストと同一キャッシュエントリのアドレス（キャッシュエントリ番号）が格納されているかチェックする（ステップ 7 0 8）。チェックの仕方は投棄リードリクエストの場合と同様である。もしあれば、投棄リードデータバッファ 4 0 3 ~ 4 0 6 の該当エントリからデータを読み出し、セクタ 4 1 3 ~ 4 1 6、セクタ 4 1 2、バス 4 0 7 を介し、リプライデータとしてバス 1 3 にデータを送りだす（ステップ 7 1 0）。即ち、リクエストコントローラ 4 0 0 は、バス 4 2 2 に投棄リードリクエストバッファ側の選択信号を出力し、バス 4 2 1 にリードリクエストに含まれるヒットウェイ番号を選択信号として出力する。これにより、投棄リードデータバッファ 4 0 3 ~ 4 0 6 の該当エントリから読み出された 4 ウェイ分のデータが、まずセクタ 4 1 3 ~ 4 1 6 で選択され、次に、該 4 ウェイ中のヒットウェイ番号に対応するデータがセクタ 4 1 2 で選択され、リプライデータとしてバス 4 0 7 を介してバス 1 3 に送出される。その後、投棄リードリクエストバッファ 4 0 1 の該当エントリを無効化する（ステップ 7 1 1）。

【 0 0 3 3 】

もし投棄リードリクエストバッファ 4 0 1 にリードリクエストと同一キャッシュエントリのアドレスがない場合、リクエストコントローラ 4 0 0 では、バス 2 1 8、3 0 を介してキャッシュデータ部 7 に当該リードリクエストを転送し、キャッシュデータ部 7 の該当キャッシュエントリから読み出された 4 ウェイ分のキャッシュデータをバス 4 0 8 ~ 4 1 1 を介してセクタ 4 1 3 ~ 4 1 6 で選択し、そのうちのヒットウェイ番号に対応するデータをセクタ 4 1 2 で選択して、リプライデータとしてバス 2 0 7 からバス 1 3 に送り出す（ステップ 7 1 2）。なお、このケースは、投棄リードリクエストによりキャッシュデータ部 7 から投棄リードデータバッファ 4 0 3 ~ 4 0 6 に先き読みしたデータが、その後の対応するリードリクエストの前に次に説明するようなライトリクエスト（先行ライトリクエスト）により無効化されていた場合に発生する。

【0034】

コヒーレントコントローラ20からパス12、217を介して受け取ったリクエストが投棄リードリクエストでもリードリクエストでもない場合、すなわちライトリクエストの場合も、投棄リードリクエストバッファ401に同一キャッシュエントリへのアドレスが格納されているかチェックする（ステップ713）。もしあれば、投棄リードリクエストバッファ401の該当エントリを無効化する（ステップ714）。次に、パス218、30を介してキャッシュデータ部7にライトリクエストを送出し、同時にコヒーレントコントローラ20からパス13を介して受け取ったキャッシュデータをバス207、バス427、バス208～211を介してバス31に送出し、キャッシュデータ部7の指定されたエントリの指定されたウエイ番号に該データを書き込む（ステップ715）。

【0035】

図9中、ステップ714において、コヒーレントコントローラ20から受け取ったライトリクエストと同一エントリへのリクエストが投棄リードリクエストバッファ401内にある場合に、該当エントリを無効化するのは、ライトによってキャッシュデータ部7内のデータが書き換えられてしまい、投棄リードデータバッファ403～406内のデータと不一致となってしまうためである。このステップ714の無効化処理により、同一キャッシュエントリに対する後続のリードリクエストでは、ステップ712において、キャッシュデータ部7から書き換えられた新しいデータが読み出されることになる。

【0036】

本実施の形態では、図9に示すように、キャッシュデータコントローラ6は、コヒーレントコントローラ20から受け取ったリードリクエストが、投棄リードリクエストバッファ401内のリクエストと同一エントリへのリクエストであった場合、キャッシュデータ部7からではなく、投棄リードデータバッファ403～406に先読みして格納されているデータを選択して返すことにより、キャッシュデータ部7のアクセスレイテンシを削減することができる。したがって、図5に示すように、コヒーレントコントローラ20では、キャッシュヒットチェックを行っている間に、投棄リードを発行しておけば、キャッシュヒットチェック

時間分のサイクルをメモリアクセスレイテンシから削減することが可能となる。

【0037】

図10および図11は本発明の他の実施形態におけるコヒーレントコントローラ20とキャッシュデータコントローラ6内のリクエストコントローラ400の処理フローを示したものである。

【0038】

図10はコヒーレントコントローラ20の処理フローである。図10が先の図5と異なる点は、ステップ800が追加されたことである。ステップ800は、ステップ302でキャッシュデータコントローラ6に発行した投棄リードリクエストがキャッシュミスだった場合、キャッシュデータコントロール6に対して、当該投棄リードリクエストで先読みした投棄リードデータを無効化するリクエスト（投棄リードデータ破棄リクエスト）を発行するものである。これにより、キャッシュデータコントロール6では、投棄リードデータバッファ402内の使用されない投棄リードデータを無効化することができるため、投棄リードリクエストバッファ401及び投棄リードデータバッファ402の有効利用が可能になる。

【0039】

図11はキャッシュデータコントローラ6内のリクエストコントローラ400の処理フローである。図11が先の図9と異なる点は、ステップ900と901が追加されたことである。このステップ900と901がコヒーレントコントローラ20から投棄リードデータ破棄リクエストを受け付けた場合の処理フローである。即ち、リクエストコントローラ400では、コヒーレントコントローラ20から投棄リードデータ破棄リクエストを受け取ったならば（ステップ900）、投棄リードリクエストバッファ401内の当該投棄リードデータ破棄リクエストに対応する投棄リードリクエストのキャッシュエントリ番号が登録されているエントリを無効化する（ステップ901）。これにより、投棄リードリクエストバッファ401及び投棄リードデータバッファ402の有効利用が可能になり、該バッファ401、402のエントリ数にある程度余裕をもって構成すると、フル状態を解消し、ステップ703、704のフル制御自体を不要とすることも可

能になる。

【0040】

本実施の形態においても、先の実施の形態1と同様に、キャッシュデータコントローラ6では、コヒーレントコントローラ20から受け取ったリードリクエストが、投棄リードリクエストバッファ401内の投棄リードリクエストと同一エントリへのリクエストであった場合、キャッシュデータ部7からではなく、投棄リードデータバッファ403～406のいずれかからデータを読み出すことにより、キャッシュデータ部7のアクセスレイテンシを削減することができる。したがって、コヒーレントコントローラ20においてキャッシュヒットチェックを行っている間に、投棄リードリクエストを発行しておけばキャッシュヒットチェック時間分のサイクルをメモリアクセスレイテンシから削減することが可能となる。

【0041】

以上、本発明の実施の形態では、キャッシュは4ウェイセットアソシアティブであるとしたが、一般にウェイ数は1以上いくつでもよく、また、本発明は、セットアソシアティブキャッシュに限らず、キャッシュタグ部分とキャッシュデータ部を別LSI等で管理するキャッシュ方式を採用したコンピュータシステムに広く適用可能であることは言うまでもない。

【0042】

【発明の効果】

以上説明したように、本発明によれば、大容量キャッシュ等を実現するため、キャッシュタグ部分とキャッシュデータ部分を別LSI等で管理した場合のキャッシュデータの読み出し時間を短縮することが可能になる。

【図面の簡単な説明】

【図1】

本発明の一実施形態のコンピュータシステムを示すブロック図である。

【図2】

CPUからのアドレスとキャッシュタグ、キャッシュエントリ番号の関係を示す図である。

【図 3】

キャッシュタグ部の構成例を示す図である。

【図 4】

キャッシュデータ部の構成例を示す図である。

【図 5】

本発明の第 1 の実施形態によるコヒーレントコントローラの処理フロー図である。

【図 6】

キャッシュデータコントローラの詳細ブロック図である。

【図 7】

キャッシュデータコントローラ内の投棄リードリクエストバッファの構成例を示す図である。

【図 8】

キャッシュデータコントローラ内の投棄リードデータバッファの構成例を示す図である。

【図 9】

本発明の第 1 の実施形態によるキャッシュデータコントローラ内のリクエストコントローラの処理フロー図である。

【図 1 0】

本発明の第 2 の実施形態によるコヒーレントコントローラの処理フロー図である。

【図 1 1】

本発明の第 2 の実施形態によるキャッシュデータコントローラ内のリクエストコントローラの処理フロー図である。

【図 1 2】

従来の 4 ウエイセットアソシアティブキャッシュの構成例を示す図である。

【符号の説明】

1, 2 CPU

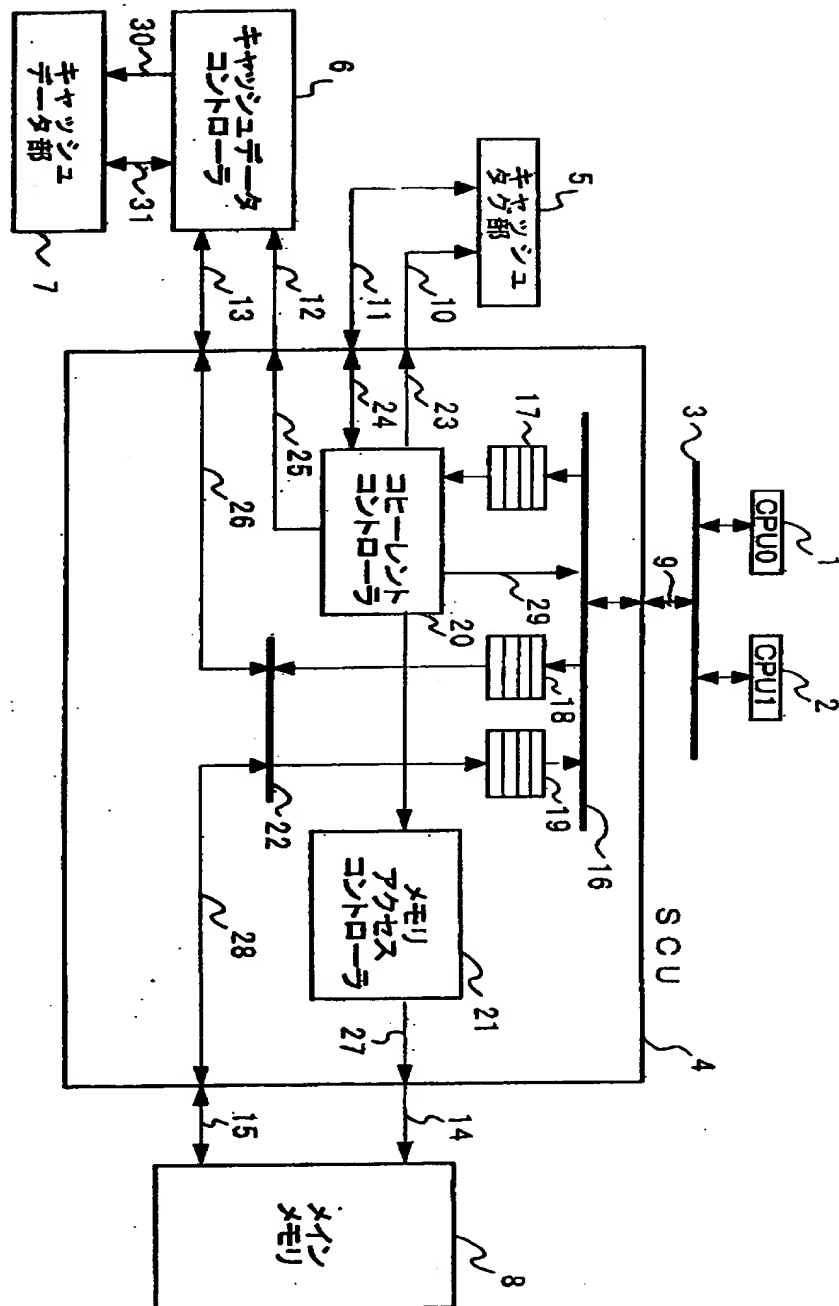
5 キャッシュタグ部

- 6 キャッシュデータコントローラ
- 7 キャッシュデータ部
- 8 メインメモリ
- 20 コヒーレンスコントローラ
- 21 メモリアクセスコントローラ
- 400 リクエストコントローラ
- 401 投棄リードリクエストバッファ
- 402 アドレスコンパレータ
- 403～406 投棄リードデータバッファ

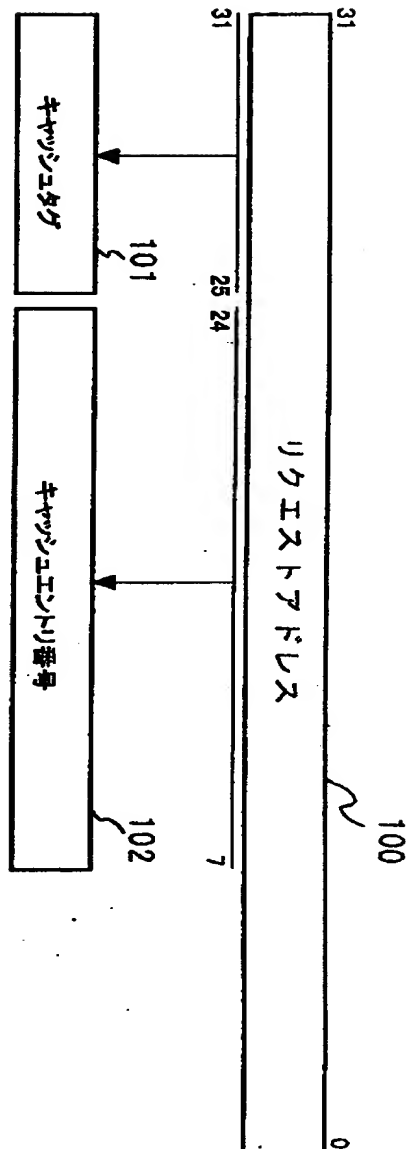
【書類名】

図面

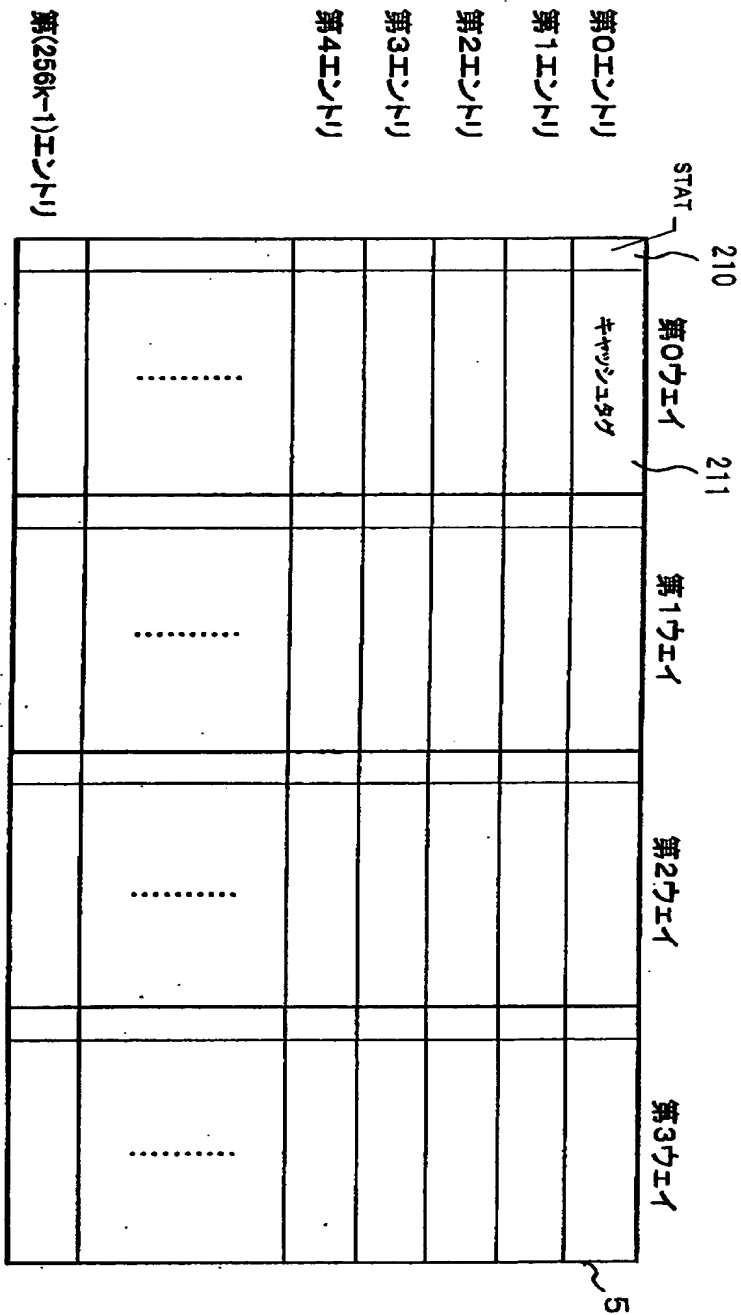
【図 1】



【図 2】



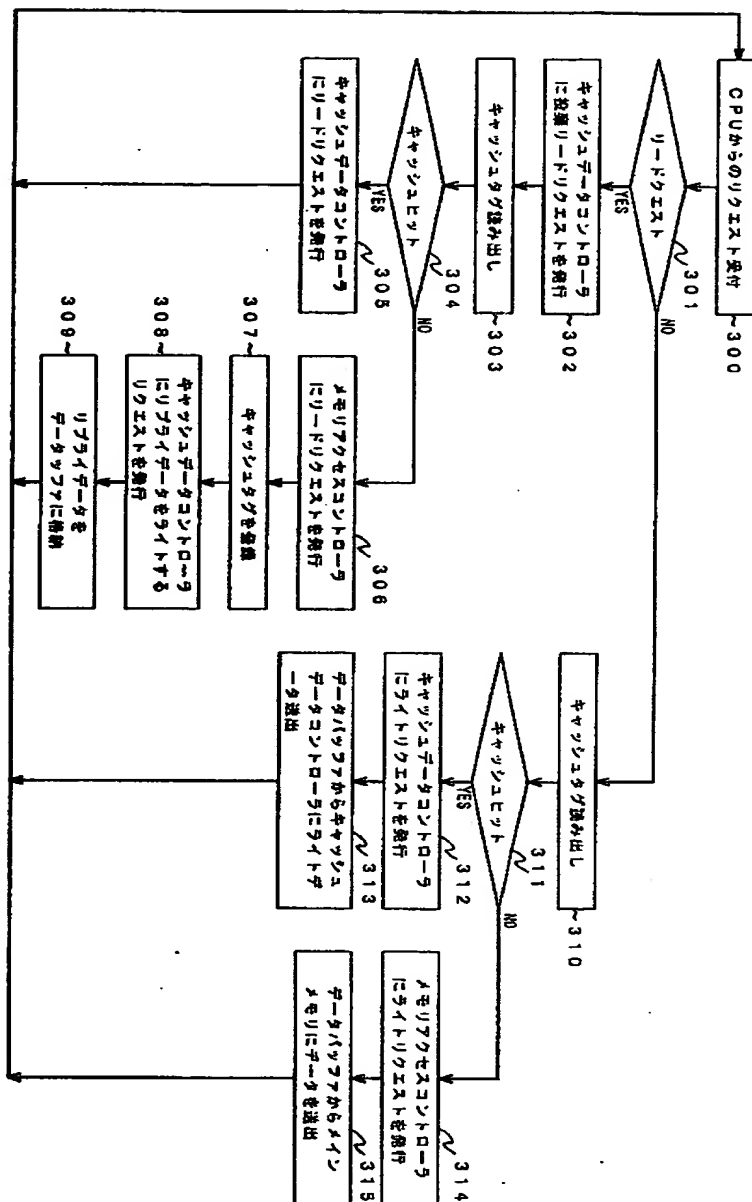
【図 3】



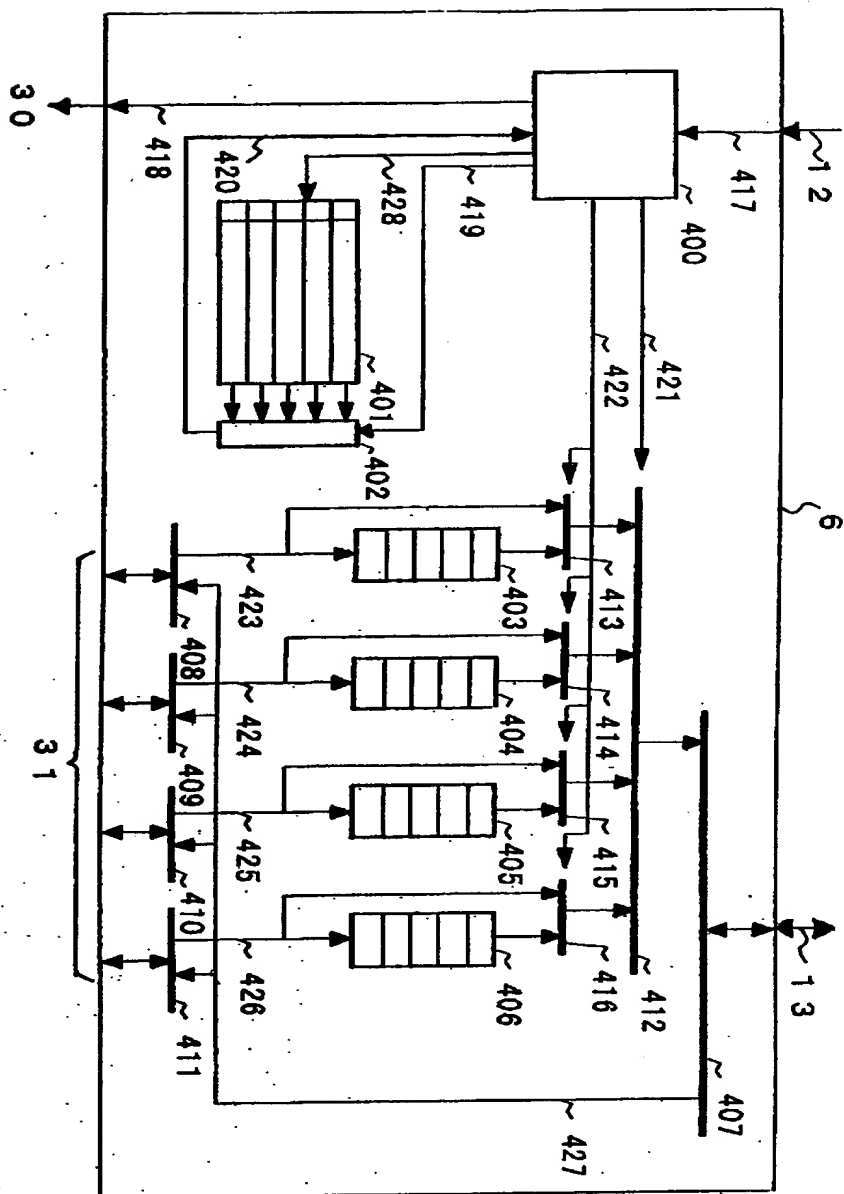
【図4】

	第0エンティ	第1エンティ	第2エンティ	第3エンティ
第0エンティ	キヤリシデータ(32B)			
第1エンティ				
第2エンティ				
第3エンティ				
第4エンティ				
第(256K-1)エンティ

【図 5】



【図 6】



【図 7】

0	V	キャッシュエントリ番号
1		
2		
	⋮	⋮
m-1		

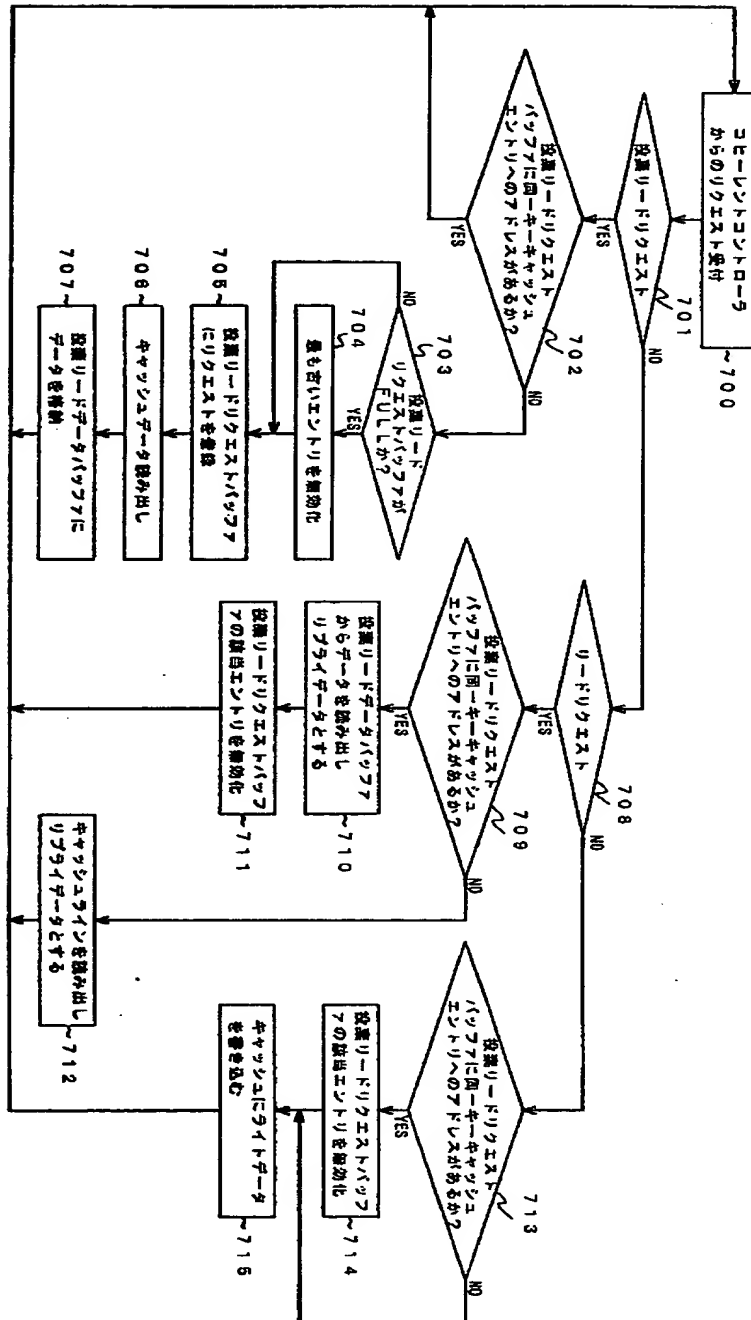
500 501 401

【図 8】

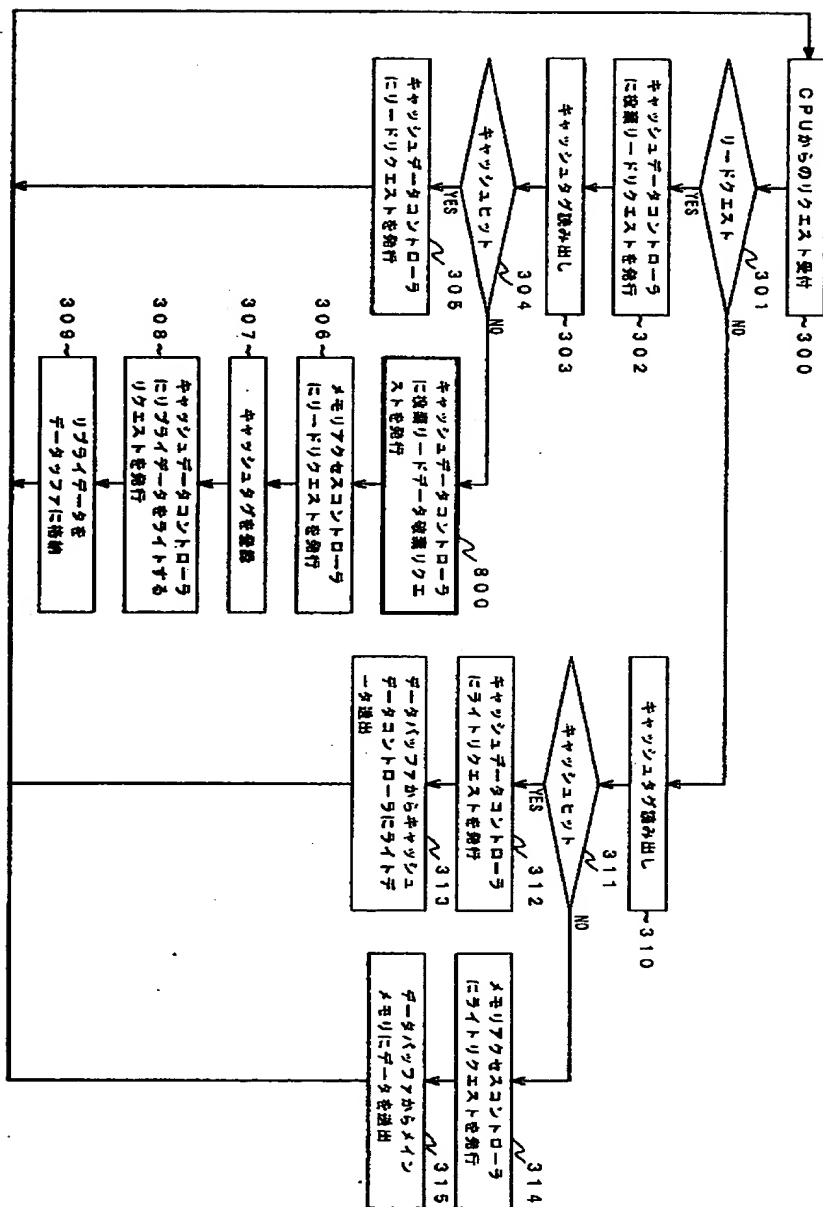
0	キャッシュデータ(32B)
1	
2	
	⋮
m-1	

600 403~406

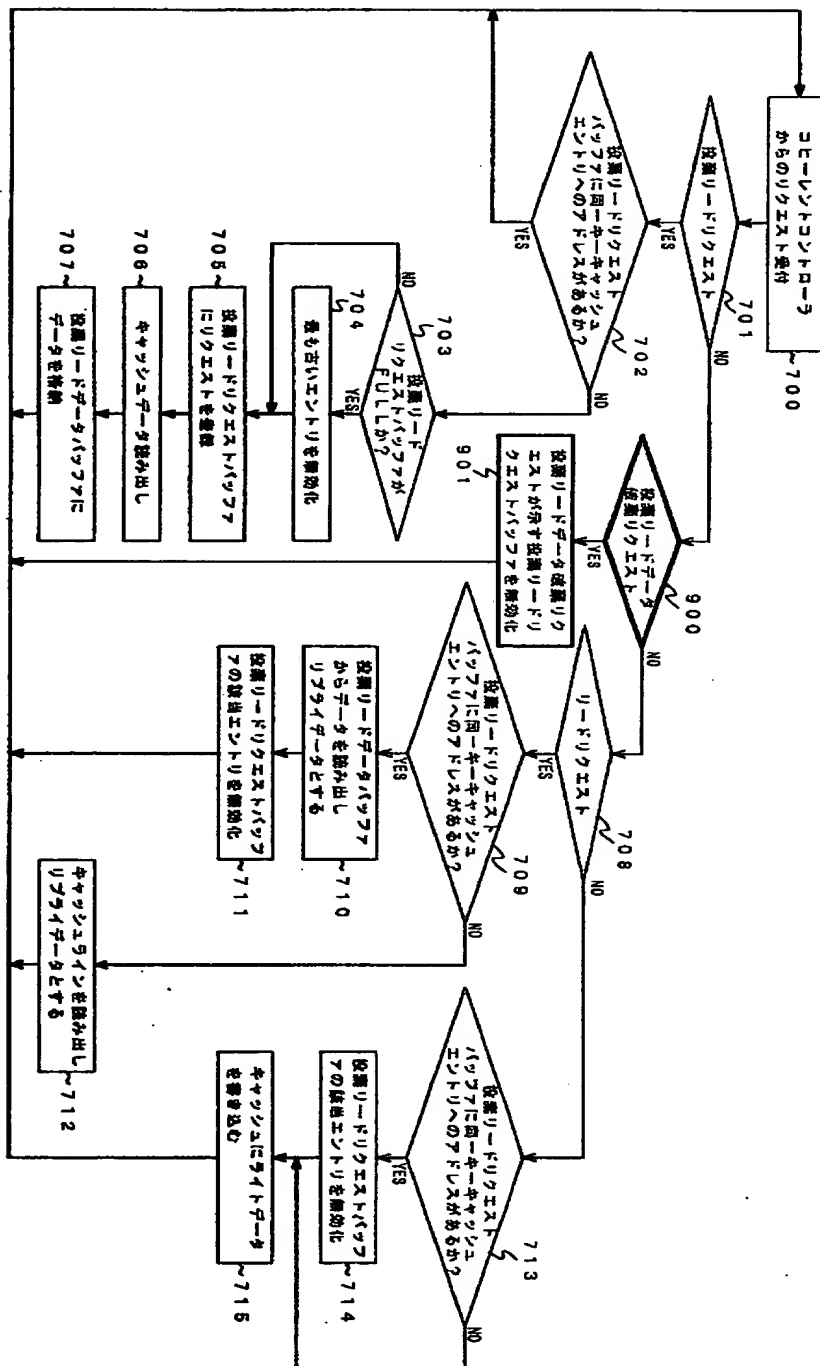
【図 9】



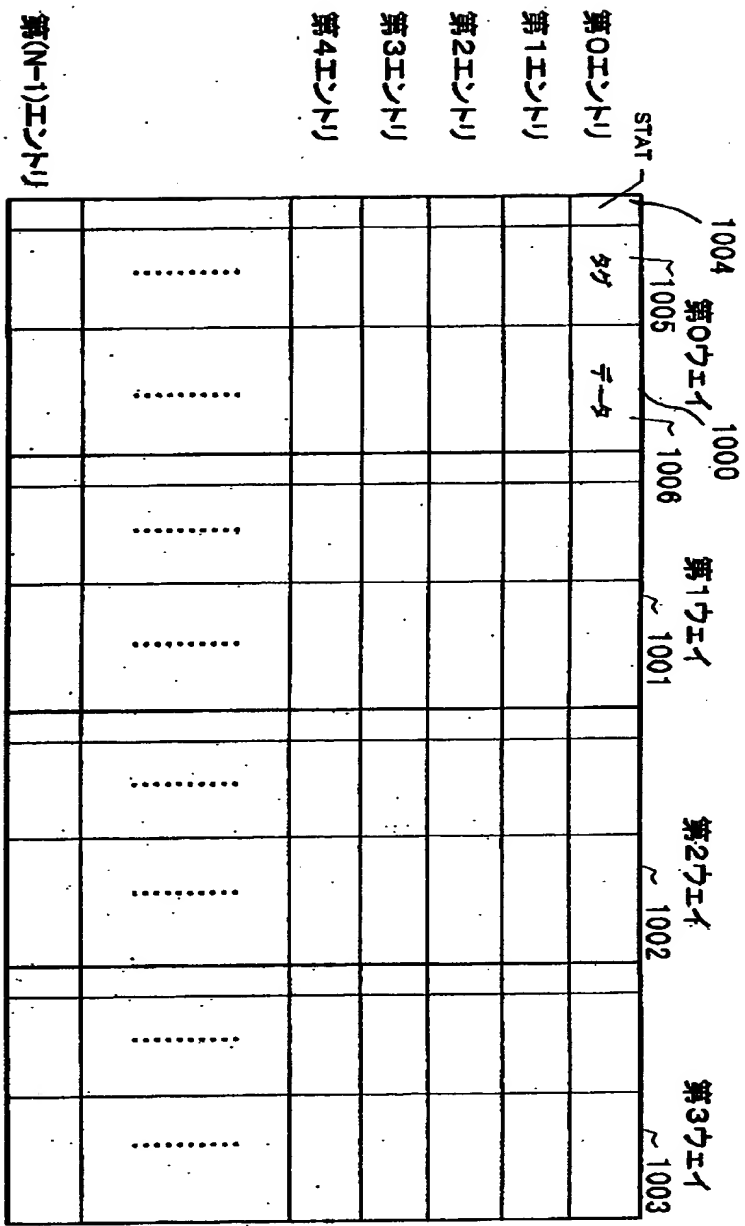
【図10】



【図11】



【図 1 2】



【書類名】 要約書

【要約】

【課題】 キャッシュ方式のコンピュータシステムにおいて、キャッシュタグ部とキャッシュデータ部が分割管理された場合のキャッシュデータの読み出しレイテンシを削減する。

【解決手段】 コヒーレントコントローラ 20 は、メモリリード処理の場合、キャッシュタグ部 5 からキャッシュタグを読み出してキャッシュヒットチェックを行う前に、キャッシュデータコントローラ 6 に対して、キャッシュデータ部 7 からデータを先読み（投棄リード）しておく投棄リードリクエストを発行し、キャッシュデータコントローラ 6 では、キャッシュデータ部 7 から投棄リードしたデータを保持しておき、キャッシュヒットした場合にコヒーレントコントローラ 20 から発行されるリードリクエストを受け取った時点で、投棄リードしておいたデータをリプライデータとして返送する。

【選択図】 図 1

出 願 人 履 歴 情 報

識別番号 [0 0 0 0 0 5 1 0 8]

1. 変更年月日 1 9 9 0 年 8 月 3 1 日

[変更理由] 新規登録

住 所 東京都千代田区神田駿河台 4 丁目 6 番地
氏 名 株式会社日立製作所